Azucar: A SAT-Based CSP Solver Using Compact Order Encoding (Tool Presentation)

Tomoya Tanjo[1], Naoyuki Tamura[2], and Mutsunori Banbara[2]

[1] Transdisciplinary Research Integration Center, Japan
[2] Information Science and Technology Center, Kobe University, Japan
tanjo@nii.ac.jp, {tamura,banbara}@kobe-u.ac.jp

Abstract. This paper describes a SAT-based CSP solver Azucar. Azucar solves a finite CSP by encoding it into a SAT instance using the compact order encoding and then solving the encoded SAT instance with an external SAT solver. In the compact order encoding, each integer variable is represented by using a numeral system of base $B \geq 2$ and each digit is encoded by using the order encoding. Azucar is developed as a new version of an award-winning SAT-based CSP solver Sugar. Through some experiments, we confirmed Azucar can encode and solve very large domain sized CSP instances which Sugar can not encode, and shows better performance for Open-shop scheduling problems and the Cabinet problems of the CSP Solver Competition benchmark.

1 Introduction

A (finite) Constraint Satisfaction Problem (CSP) is a combinatorial problem to find an assignment which satisfies all given constraints on finite domain variables [1]. A SAT-based CSP solver is a program which solves a CSP by encoding it to SAT [2] and searching solutions by SAT solvers.

There have been several SAT-based CSP solvers developed, such as Sugar [1] [3], FznTini [4], SAT4J CSP [5], and others. Especially, Sugar became a winner of several categories at the recent International CSP Solver Competitions in two consecutive years. It uses order encoding [6] which shows good performance on various applications [6–8] and is known as the only SAT encoding reducing tractable CSP to tractable SAT [9]. In the order encoding, the Unit Propagation in SAT solvers corresponds to the Bounds Propagation in CSP solvers.

In this paper, we describe a SAT-based CSP solver Azucar [2]. It uses a new SAT encoding method named compact order encoding [10–12], in which each integer variable is divided into digits by using a numeral system of base $B \geq 2$ and each digit is encoded by using the order encoding. Therefore, it is equivalent to the order encoding [6] when $B \geq d$ and it is equivalent to the log encoding [13]

[1] http://bach.istc.kobe-u.ac.jp/sugar/
[2] http://code.google.com/p/azucar-solver/

when $B = 2$ where d is the maximum domain size. In that sense, the compact order encoding is the generalization and integration of both encodings.

Size of generated SAT instances by the compact order encoding using two or more digits are much smaller than those by the direct [14], support [15], and order encodings. Therefore, it is another encoding applicable to large domain sized CSPs besides the log and log-support [16] encodings. In the compact order encoding, the Unit Propagation in SAT solvers corresponds to the Bounds Propagation in the most significant digit in CSP solvers. Therefore, the conflicts are likely to be detected with fewer decisions than the log and log-support encodings. These observations are confirmed through the experimental results on Open-Shop Scheduling problems with large domain sizes in which the compact order encoding is about 5 times faster than the log encoding on average.

Azucar is a first implementation of the compact order encoding and it is developed as an enhancement version of Sugar. User can specify either the number of digits m or the base B as the command line option of Azucar. When $m = 1$ is specified, Azucar uses the order encoding to encode the given CSP. The log encoding is used when $B = 2$ is specified. If user specifies neither m nor B, Azucar uses $m = 2$ by default. In various problems, Azucar with $m \in \{2, 3\}$ shows the better performance than Sugar especially for large domain sized CSP.

2 Compact Order Encoding

The basic idea of the compact order encoding is the use of a numeral system of base $B \geq 2$ [10–12]. That is, each integer variable x is represented by a summation $\sum_{i=0}^{m-1} B^i x^{(i)}$ where $m = \lceil \log_B d \rceil$ and $0 \leq x^{(i)} < B$ for all integer variables $x^{(i)}$, and each $x^{(i)}$ is encoded by using the order encoding. As described in Section 1, the compact order encoding is equivalent to the order encoding [6] when $B \geq d$ and it is equivalent to the log encoding [13] when $B = 2$.

In this paper, we will show some examples to encode an integer variable and a constraint by using the compact order encoding. More details about the compact order encoding are described in [10–12].

For example, when we choose $B = 3$, the integer variable $x \in \{0..8\}$ is divided into two digits as follows. Each $x^{(i)}$ represents the i-th digit of x and $x^{(1)}$ represents the most significant digit.

$$x^{(1)}, x^{(0)} \in \{0..2\}$$

By using the order encoding, these propositional variables are introduced where $p(x^{(i)} \leq a)$ is defined as true if and only if the comparison $x^{(i)} \leq a$ holds. $p(x^{(1)} \leq 2)$ is not necessary since $x^{(1)} \leq 2$ is always true.

$$p(x^{(1)} \leq 0) \quad p(x^{(1)} \leq 1)$$
$$p(x^{(0)} \leq 0) \quad p(x^{(0)} \leq 1)$$

To represents the order of propositional variables, these two clauses are required. For instance, $\neg p(x^{(1)} \leq 0) \vee p(x^{(1)} \leq 1)$ represents $x^{(1)} \leq 0 \Rightarrow x^{(1)} \leq 1$.

$$\neg p(x^{(1)} \leq 0) \vee p(x^{(1)} \leq 1) \qquad \neg p(x^{(0)} \leq 0) \vee p(x^{(0)} \leq 1)$$

Constraint	Order Encoding	Compact Order Encoding	Log Encoding
$x \le a$	$O(1)$	$O(m)$	$O(\log_2 d)$
$x \le y$	$O(d)$	$O(mB)$	$O(\log_2 d)$
$z = x + a$	$O(d)$	$O(mB)$	$O(\log_2 d)$
$z = x + y$	$O(d^2)$	$O(mB^2)$	$O(\log_2 d)$
$z = xy$	$O(d^2)$	$O(mB^3 + m^2B^2)$	$O(\log_2^2 d)$

Fig. 1. Comparison of different encodings on the number of SAT-encoded clauses

Each constraint is divided into digit-wise constraints and then encoded by using the order encoding. For example, a constraint $x \le y$ $(x, y \in \{0..8\})$ is encoded into the following clauses where p is a new propositional variable which represents $\neg(x^{(0)} \le y^{(0)})$. C_0 and C_1 represent $x^{(1)} \le y^{(1)}$, C_2, C_3 and C_4 represent $p \rightarrow x^{(1)} \le y^{(1)} - 1$, and C_5 and C_6 represent $\neg(x^{(0)} \le y^{(0)}) \rightarrow p$.

$$\begin{aligned}
C_0 &: \quad p(x^{(1)} \le 0) \vee \neg p(y^{(1)} \le 0) \\
C_1 &: \quad p(x^{(1)} \le 1) \vee \neg p(y^{(1)} \le 1) \\
C_2 &: \quad \neg p \vee \neg p(y^{(1)} \le 0) \\
C_3 &: \neg p \vee p(x^{(1)} \le 0) \vee \neg p(y^{(1)} \le 1) \\
C_4 &: \quad \neg p \vee p(x^{(1)} \le 1) \\
C_5 &: p \vee p(x^{(0)} \le 0) \vee \neg p(y^{(0)} \le 0) \\
C_6 &: p \vee p(x^{(0)} \le 1) \vee \neg p(y^{(0)} \le 1)
\end{aligned}$$

Let d be the domain size of integer variables, B be the base and $m = \lceil \log_B d \rceil$ be the number of digits. Fig. 1 shows the number of clauses required to encode each constraint. In the compact order encoding, each addition $z = x + y$ and multiplication $z = xy$ are encoded into $O(mB^2)$ and $O(mB^3 + m^2B^2)$ clauses respectively. It is much less than $O(d^2)$ clauses of the order encoding and thus it can be applicable to large domain CSP.

We also show the relations between the Unit Propagation in SAT solvers in each encodings and the constraint propagation in CSP solvers. In the order encoding, the Unit Propagation in SAT solvers corresponds to the Bounds Propagation in CSP solvers. The compact order encoding can achieve the Bounds Propagation in the most significant digit while the log encoding achieves the Bounds Propagation in the most significant bit. Therefore the compact order encoding can detect the conflicts earlier and thus it can solve CSP faster than the log encoding.

3 Azucar Implementation

Azucar is an open-source SAT-based CSP solver distributed under the BSD 3-clause license. Azucar encodes a CSP into SAT by using the compact order encoding, and then the SAT-encoded instance are solved by an external SAT solver such as MiniSat [17], SAT4J [18] or GlueMiniSat [3]. Azucar can handle finite CSP over integers written in Lisp-like input format or XCSP 2.1 format [4] which is used in the 2009 International CSP Solver Competition. Azucar can receive one of these options where d is the maximum domain size of integer variables.

- -b B: Azucar uses the numeral system of base B (i.e. $m = \lceil \log_B d \rceil$).
- -m m: Azucar divides each integer variable into at most m digits (i.e. $B = \lceil \sqrt[m]{d} \rceil$).

The encoder and decoder are written in Java, and the frontend of Azucar is written in Perl.

4 Performance Evaluation

To evaluate the scalability and efficiency of our encoding used in Azucar, we used 85 Open-Shop Scheduling problems with very large domain sizes, which are generated from "j7" and "j8" by Brucker *et al.* by multiplying the process times by some constant factor c. The factor c is varied within 10^i ($i \in \{0, 1, 2, 3, 4\}$). For example, when $c = 10^4$, the maximum domain size d becomes about 10^7.

We compare four different encodings: the order encoding which is used in Sugar, the compact order encoding with $m \in \{2, 3\}$, and the log encoding. For each instance, we set its makespan to the optimum value minus one and then encode it into SAT. Such SAT-encoded instances are unsatisfiable. We use the MiniSat solver [17] as a backend SAT solver.

Factor c	Domain Size d	#Instances	Order Encoding	Compact Order Encoding $m = 2$	Compact Order Encoding $m = 3$	Log Encoding
1	10^3	17	13	**14**	**14**	**14**
10	10^4	17	12	**13**	**13**	**13**
10^2	10^5	17	8	**13**	**13**	12
10^3	10^6	17	0	**14**	13	12
10^4	10^7	17	0	12	**13**	**13**
	Total	85	34	**66**	**66**	63

Fig. 2. Benchmark results of different encodings on the number of solved instances for OSS benchmark set by Brucker *et al.* with multiplication factor c

Fig. 2 shows the number of solved instances within 3600 seconds by four solvers. All times were collected on a Linux machine with Intel Xeon 3.0 GHz, 16GB Memory. "Domain size d" indicates the approximate average of domain size of integer variables. We highlight the best number of solved instances.

The compact order encoding solved the most instances for any factor c and totally 66 out of 85 instances rather than 63 by the log encoding and 34 by the

[3] `http://glueminisat.nabelab.org/`

[4] `http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf`

order encoding. The compact order encoding with $m = 3$ can be highly scalable with the growth of c compared with the order encoding. For example, when $c = 1000$, it solved 13 out of 17 instances (76%), while none (0%) by the order encoding due to the memory limitation. Moreover, it is fastest on average when $c \geq 10$. For example, it solved about 5 times faster than the log encoding when $d \approx 10^7$.

Fig. 3 shows the cactus plot of benchmark results in which the number of solved instances is on the x-axis and the CPU time is on the y-axis. The compact order encoding solved the most instances for almost any CPU time limit. For example, the compact order encoding with $m = 3$ solved 61 instances within 600 seconds while the order encoding was 25, the compact order encoding with $m = 2$ was 56, and the log encoding was 53.

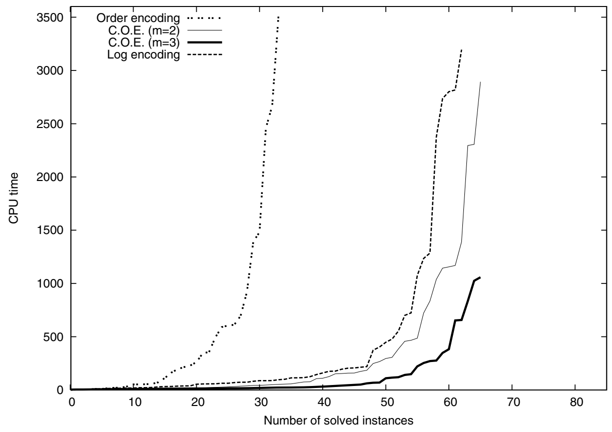

Fig. 3. Cactus plot of various encodings for 85 OSS instances

To evaluate the efficiency of our encoding for smaller domain CSP, we also used graph coloring problems published in Computational Symposium on Graph Coloring and its Generalizations [5] and we confirmed that the compact order encoding can solve the almost same number of instances compared with the order encoding even when the domain size is less than 10^2.

Finally, to evaluate the efficiency of Azucar for large domain CSP, we also used the Cabinet problems in GLOBAL category in the CSP Solver Competitions and we confirmed that Azucaris over 1.7 times faster than Sugar on average.

5 Conclusion

In this paper, we described a SAT-based CSP solver Azucar. Through some experiments, Azucar with $m \in \{2, 3\}$ shows the better performance than Sugar

[5] `http://mat.gsia.cmu.edu/COLOR04/`

especially for large domain sized CSP. Finally, although the compact order encoding used in Azucar is developed to encode CSP, it can be applicable to other problems dealing with the arithmetic constraints.

Acknowledgment. The authors would like to thank Prof. K. Inoue from the National Institute of Informatics for his valuable comments. This work is supported in part by Grant-in-Aid for Scientific Research (B) 2430000.

References

1. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier Science Inc. (2006)
2. Prestwich, S.D.: CNF encodings. In: Handbook of Satisfiability, pp. 75–97. IOS Press (2009)
3. Tamura, N., Tanjo, T., Banbara, M.: System description of a SAT-based CSP solver Sugar. In: Proceedings of the 3rd International CSP Solver Competition, pp. 71–75 (2008)
4. Huang, J.: Universal Booleanization of Constraint Models. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 144–158. Springer, Heidelberg (2008)
5. Le Berre, D., Lynce, I.: CSP2SAT4J: A simple CSP to SAT translator. In: Proceedings of the 2nd International CSP Solver Competition, pp. 43–54 (2008)
6. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints 14(2), 254–272 (2009)
7. Soh, T., Inoue, K., Tamura, N., Banbara, M., Nabeshima, H.: A SAT-based method for solving the two-dimensional strip packing problem. Fundamenta Informaticae 102(3-4), 467–487 (2010)
8. Banbara, M., Matsunaka, H., Tamura, N., Inoue, K.: Generating Combinatorial Test Cases by Efficient SAT Encodings Suitable for CDCL SAT Solvers. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 112–126. Springer, Heidelberg (2010)
9. Petke, J., Jeavons, P.: The Order Encoding: From Tractable CSP to Tractable SAT. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 371–372. Springer, Heidelberg (2011)
10. Tanjo, T., Tamura, N., Banbara, M.: A Compact and Efficient SAT-Encoding of Finite Domain CSP. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 375–376. Springer, Heidelberg (2011)
11. Tanjo, T., Tamura, N., Banbara, M.: Proposal of a compact and efficient SAT encoding using a numeral system of any base. In: Proceedings of the 1st International Workshop on the Cross-Fertilization Between CSP and SAT, CSPSAT 2011 (2011)
12. Tanjo, T., Tamura, N., Banbara, M.: Towards a compact and efficient SAT-encoding of finite linear CSP. In: Proceedings of the 9th International Workshop on Constraint Modelling and Reformulation, ModRef 2010 (2010)
13. Iwama, K., Miyazaki, S.: SAT-variable complexity of hard combinatorial problems. In: Proceedings of the IFIP 13th World Computer Congress, pp. 253–258 (1994)
14. de Kleer, J.: A comparison of ATMS and CSP techniques. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989), pp. 290–296 (1989)

15. Kasif, S.: On the parallel complexity of discrete relaxation in constraint satisfaction networks. Artificial Intelligence 45(3), 275–286 (1990)
16. Gavanelli, M.: The Log-Support Encoding of CSP into SAT. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 815–822. Springer, Heidelberg (2007)
17. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
18. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation 7(2–3), 56–64 (2010)